

A Learning Algorithm for Multi-Layer Perceptrons With Hard-Limiting Threshold Units

Rodney M. Goodman and Zheng Zeng
Department of Electrical Engineering, 116-81
California Institute of Technology
Pasadena, CA 91125

Abstract— We propose a novel learning algorithm to train networks with multi-layer linear-threshold or hard-limiting units. The learning scheme is based on the standard back-propagation, but with “pseudo-gradient” descent, which uses the gradient of a sigmoid function as a heuristic hint in place of that of the hard-limiting function. A justification that the pseudo-gradient always points in the right down hill direction in error surface for networks with one hidden layer is provided. The advantages of such networks are that their internal representations in the hidden layers are clearly interpretable, and well-defined classification rules can be easily obtained, that calculations for classifications after training are very simple, and that they are easily implementable in hardware. Comparative experimental results on several benchmark problems using both the conventional back-propagation networks and our learning scheme for multi-layer perceptrons are presented and analyzed.

I. INTRODUCTION

Single-layer networks of linear threshold units (or hard-limiting units) known as perceptrons have been shown to have very limited learning capacity [2]. Although multi-layer systems of such units are much more powerful than single-layer ones, there has been no known learning algorithm for such networks.

In recent years, networks with continuous, nonlinear activation functions have been shown to be able to perform much more complicated tasks than single-layer perceptrons. With the differentiable activation functions, gradient descent can then be used to train such networks [4].

The research described in this paper was supported by ARPA under grants number AFOSR-90-0199 and N00014-92-J-1860.

However, the internal representations of these networks have been hard to analyze, due to the fact that their activation spaces are continuous, and high dimensional. Multi-layer perceptron networks are thus still of interest. In addition to easily understandable internal representations, classification rules can be readily obtained from trained perceptron networks, the operations of the networks after being successfully trained are extremely simple, and they are easy to implement in hardware.

In this paper, we attempt to solve the problem of training multi-layer hard-limiting-unit networks by using non-zero values for logic 0's and 1's, and by a pseudo-gradient descent learning scheme. Henceforth, these networks will be called interchangeably, as discrete networks or perceptron networks throughout this paper.

II. NETWORK ARCHITECTURE

Shown in Fig. 1 is a two-layer network of hard-limiting units. Note that since the output layer is “discretized”, such networks are therefore used for classification or encoding problems. We use $S_i^{(l)}$ to denote the output value of unit i in layer l , where the 0th layer is defined to be the input layer, and $w_{ij}^{(l)}$ to denote the weight connecting from unit j in layer $l-1$ to unit i in layer l . The operational equations for the network are:

$$S_i^{(l)} = D_0\left(\sum_j w_{ij}^{(l)} S_j^{(l-1)}\right), \quad \forall l, i, \quad (1)$$

$$\text{where } D_0(x) = \begin{cases} 0.8 & \text{if } x \geq 0.0 \\ 0.2 & \text{if } x < 0.0. \end{cases} \quad (2)$$

Note that the values 0.2 and 0.8 are used here instead of 0 and 1 in order for logic “0”s to have some power of influence over the next layers. These values play an important role in the pseudo-gradient learning which is explained in the following section.

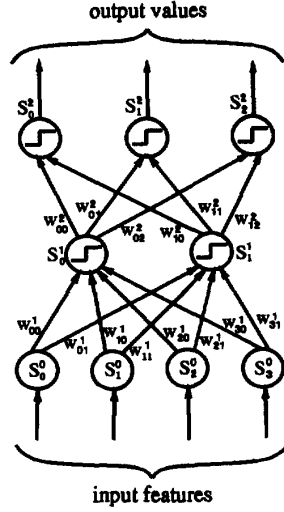


Figure 1: A network of perceptrons with a single hidden layer

III. PSEUDO-GRADIENT LEARNING AND ITS JUSTIFICATION

Our learning scheme is based on the standard back-propagation method [4], but with “pseudo-gradient” descent instead of gradient descent on the error surface. A learning method based on a similar idea for training recurrent networks was first introduced in [6, 7].

To explain the pseudo-gradient, we need to introduce another set of values for the output and hidden layers, which we will call the analog values of the units, as opposed to the discrete (hard-limited) values that are actually used during network operations:

$$h_i^{(l)} = f(\text{net}_i^{(l)}), \quad \forall l, i, \quad (3)$$

where

$$\text{net}_i^{(l)} = \sum_j w_{ij}^{(l)} S_j^{(l-1)} \quad (4)$$

and

$$f(x) = \frac{1}{1 + e^{-x}}. \quad (5)$$

From (1) to (5), it is obvious that

$$S_i^{(l)} = D(h_i^{(l)}),$$

where

$$D(x) = \begin{cases} 0.8 & \text{if } x \geq 0.5 \\ 0.2 & \text{if } x < 0.5, \end{cases}$$

For the input layer, define $h_i^{(0)} = S_i^{(0)}$ to be the i th input.

Let L be the output layer, the error function for an input pattern is defined to be:

$$E = \frac{1}{2} \sum_i (h_i^{(L)} - t_i)^2,$$

where t_i is the desired value for output unit i . For classification and encoding problems, t_i is either 0 or 1.

In a manner similar to back-propagation [4], the error “gradient” with respect to each weight is computed, but instead of the true gradient, we compute a value which we define to be the “pseudo-gradient”:

$$\frac{\partial \widetilde{E}}{\partial w_{ij}^{(l)}} = \widetilde{\delta}_i^{(l)} S_j^{(l-1)}, \quad \forall l, i, j, \quad (6)$$

where

$$\begin{aligned} \widetilde{\delta}_i^{(l)} &= \frac{\partial \widetilde{E}}{\partial \text{net}_i^{(l)}} \\ &= \begin{cases} f'(\text{net}_i^{(l)})(h_i^{(l)} - t_i) & \text{if } l = L \\ f'(\text{net}_i^{(l)}) \sum_k \widetilde{\delta}_k^{(l+1)} w_{ki}^{(l)} & \text{otherwise.} \end{cases} \end{aligned} \quad (7)$$

Here $\frac{\partial \widetilde{E}}{\partial w_{ij}^{(l)}}$ is what we call the “pseudo-gradient” of x with respect to $w_{ij}^{(l)}$.

Note that from (1), (2) and (6), by making the possible values of $S_j^{(l-1)}$ to be 0.2 and 0.8, instead of 0 and 1, the pseudo-gradient $\frac{\partial \widetilde{E}}{\partial w_{ij}^{(l)}}$ will not be reduced to 0 when $S_j^{(l-1)}$ is in the “off” (or logic 0) mode, thus the heuristic hint provided by $\widetilde{\delta}_i^{(l)}$ will not be eliminated.

Note also that had we computed the true gradients, the only thing that would have been different in the pseudo-gradient formulae (6) and (7) is that the term $f'(\text{net}_i^{(l)})$ in the “otherwise” case in (7) should have been $D'_0(\text{net}_i^{(l)})$. However, $D'_0(x)$ is zero everywhere and non-existent at $x = 0$. By using f' instead of D'_0 , we provide in essence a heuristic hint of which direction in x a step up (or down) of $D_0(x)$ is, and also of how far away it is from x .

Consider the case of a single-hidden-layer network. Since for the “output layer” case, i.e., $l = L = 2$, the pseudo-gradient is in fact the same as the true gradient, the “inaccuracy” of the pseudo-gradient only exists in one layer, that is, the hidden layer ($l = 1$), thus in the “otherwise” case in (7), $\widetilde{\delta}_k^{(l+1)}$ is the

| # of hidden units | discrete networks | | conventional backprop | |
|-------------------|----------------------|-----------------|-----------------------|-----------------|
| | # of successful runs | avg # of epochs | # of successful runs | avg # of epochs |
| 2 | 5 | 5000 | 3 | 4119 |
| 3 | 10 | 2920.9 | 10 | 1154.4 |
| 4 | 10 | 1801.5 | 10 | 642.6 |

Table 1: Comparative results on the binary XOR problem. All networks have 2 input and 1 output units. Both the training and test data set contain all 4 instances of XOR. The learning rate is 0.5, with no momentum term and no weight decay. Error tolerance is 0.0000001, maximum number of iterations is 5000. The “number of successful runs” is obtained out of 10 runs with different random weight initializations. The “average number of epochs” is the averages over the successful runs.

| # of hidden units | momentum | weight decay factor | discrete networks | | conventional backprop | |
|-------------------|----------|---------------------|---------------------|--------------------|-----------------------|--------------------|
| | | | average performance | standard deviation | average performance | standard deviation |
| 2 | 0.5 | 1.0 | 92.0 | 4.99 | 96.0 | 4.42 |
| 3 | 0.0 | 1.0 | 96.7 | 5.37 | 97.3 | 4.42 |
| 3 | 0.5 | 1.0 | 96.0 | 6.11 | 96.7 | 5.37 |
| 3 | 0.0 | .99 | 95.3 | 6.67 | 94.7 | 4.99 |
| 3 | 0.5 | .99 | 96.0 | 5.33 | 97.3 | 4.42 |
| 4 | 0.0 | 1.0 | 96.7 | 5.37 | 94.7 | 6.53 |
| 4 | 0.5 | 1.0 | 96.0 | 5.33 | 94.7 | 5.81 |
| 4 | 0.5 | .99 | 94.0 | 6.96 | 97.3 | 4.42 |

Table 2: Comparative results on the iris data classification problem. All networks have 4 input and 3 output units. The learning rate is 0.5, with different momentum and weight decay factors as shown. Error tolerance is 0.0000001, maximum number of iterations is 5000. The data set of 150 is randomly partitioned into 10 subsets, each of size 15. For each set of network parameters, 10 runs are made by leaving out each one of the subsets as the test set, and using the remaining 9 subsets as the training set. Performance is averaged over the 10 runs.

true $\delta_k^{(l+1)}$ from straight back-propagation. Therefore, $\sum_k \delta_k^{(l+1)} w_{kj}^{(l)}$ gives us the true value of $\sum_k \delta_k^{(l+1)} w_{kj}^{(l)}$, and since $f'(net_i^{(l)})$ is always positive, $\delta_i^{(l)}$ truly gives us a good indication of the direction, distance or size of a step up (or down) in the discontinuous error surface E as a function of $net_{ij}^{(l)}$, as does $\frac{\partial E}{\partial w_{ij}^{(l)}}$ give a similarly good indication in E as a function of $w_{ij}^{(l)}$.

IV. EXPERIMENTAL RESULTS

Shown in Tables 1 through 4 are comparative experimental results of using both the proposed discrete network training method and the standard back-propagation on the following bench mark problems, respectively: exclusive or, iris data classification [1], sonar data classification [3] and NETtalk [5]. All experiments are done with two-layer networks. Detailed

parameters are described in the corresponding captions.

The training set of the XOR problem consists of all 4 examples of the binary XOR problem. 10 runs are done with different random weight initializations for each network configuration and each of the learning schemes. In this experiment, we intend to compare the convergence speeds of the two methods. A successful run is defined to be such that the network converged within the given maximum number of epochs (in this case, 5000) during training and gives correct outputs for all 4 examples. Note that for networks with 2 hidden units, there are unsuccessful runs for both learning schemes, which means that each of the corresponding networks reached a local minimum, instead of a global one. The number of unsuccessful runs for the two are comparable: 5 for our method, and 7 for standard back-propagation.

| # of hidden units | discrete networks | | conventional backprop | |
|-------------------|---------------------|--------------------|-----------------------|--------------------|
| | average performance | standard deviation | average performance | standard deviation |
| 2 | 73.08 | 11.60 | 82.69 | 8.55 |
| 3 | 72.60 | 8.33 | 85.58 | 6.66 |
| 6 | 80.77 | 7.93 | 85.58 | 6.19 |
| 12 | 85.10 | 9.02 | 86.06 | 6.08 |
| 24 | 86.06 | 7.00 | 82.21 | 8.79 |
| 36 | 83.17 | 7.10 | 82.69 | 10.73 |
| 48 | 78.85 | 9.35 | 71.63 | 20.89 |
| 60 | 77.88 | 11.91 | 56.73 | 21.44 |

Table 3: Comparative results on the sonar data set. All networks have 60 input and 2 output units. The learning rate is 0.1 for discrete networks, and 0.2 for conventional backprop, with no momentum term and no weight decay. Error tolerance is 0.001, maximum number of iterations is 300. The data set of 208 is randomly partitioned into 13 subsets, each of size 16. For each set of network parameters, 13 runs are made by leaving out each one of the subsets as the test set, and using the remaining 12 subsets as the training set. Performance is averaged over the 13 runs.

| # of hidden units | discrete networks | | conventional backprop | |
|-------------------|-----------------------------|-------------------------|-----------------------------|-------------------------|
| | performance on training set | performance on test set | performance on training set | performance on test set |
| 15 | 77.05 | 68.41 | 83.72 | 72.64 |
| 30 | 84.53 | 71.74 | 89.72 | 75.82 |
| 80 | 90.22 | 72.55 | 93.65 | 75.90 |
| 120 | 91.95 | 73.62 | 92.52 | 75.61 |

Table 4: Comparative results on the NETtalk data set. All networks have 135 input and 22 output units. The learning rate is 0.1, with the momentum factor being 0.9 and no weight decay. Error tolerance is 0.001, maximum number of iterations is 1000. The training set consists of 1000 most commonly used words, with 5603 letters to pronounce in total. The test set consists of about 4000 words, with 19994 letters to pronounce in total.

The iris data set consists of 3 classes of 50 instances each, where each class refers to a type of iris plant. Attributes are different measurements of the flowers. 10 runs are done by partitioning the data set and using the subsets in a manner similar to cross-validation. In this experiment, we aim at investigating and comparing the effects of momentum and weight decay factors on the two learning schemes.

The sonar data set was used originally by Gorman and Sejnowski in their study of the classification of sonar signals using a neural network [3]. The task is to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock. There are 208 patterns in total with 111 belonging to the "metal" class, and 97 belonging to the "rock" class. Again, for each network configuration,

13 runs are done, in a similar manner to the iris data experiment. The purpose of this experiment is to compare the performances of the two network structures with different numbers of hidden units. The network configurations of the first 5 rows in Table 3 are the same as in [3], while the last 3 rows are additional experiments we did to obtain a comparison over a wider range.

The task of the NETtalk problem is to train a network to learn to convert English text to speech. Inputs are windows of 5 letters, with the letter to be pronounced in the center. Desired outputs are encoded phonemes. Each input letter is unary encoded by a group of 27 units. The training set consists of 1000 most commonly used words. The test set consists of about 4000 words. In this case, the problem is of a

particularly large size: 135 input, 22 output, and 15 to 120 hidden units, about 5600 training examples, and close to 20,000 test examples. We used this problem to test the performance of our network on very large problems.

V. DISCUSSION

It can be seen that in general, the performances of the proposed discrete network are comparable to those of the conventional back-propagation network on all the benchmark problems.

From the results on the XOR problem, it is clear that the pseudo-gradient training takes longer than the conventional back-propagation, due to the inaccuracies introduced for gradient descent. However, we should note that the operations needed for one epoch of training is almost the same for pseudo-gradient as back-propagation, the only difference being the discretization operations. The experiments on all the other larger data sets were done for the same fixed number of epochs (300 to 5000) for both networks, so the comparative results shown in Tables 2 to 4 are in fact of training both networks for about the same time period.

The iris data set results indicate that adding a momentum term helps to improve the performance of the discrete network but has an opposite effect on the performance of the conventional back-propagation network. On the other hand, weight decay helps to improve the performance of the conventional network but has an opposite effect on the discrete network. The reason for the phenomena is still under investigation.

For the sonar data experiment, it is expected that the performance of either of the network structure goes up with the increase of the number of hidden units, and drops after a peak has been reached. Note that it takes more hidden units for the discrete network to reach the same optimum performance as that of the conventional back-propagation network. The reason for this can be that the internal representation capacity of a discrete network is much less than that of an analog network, the former having only two possible values for each unit, and the latter having infinite values theoretically. On the other hand, for the same reason, it also takes more hidden units for the performance of the former to drop, after the optimum performance is reached, to the same level as that of the latter. That is, the discrete network overfits more slowly than the back-propagation network. Thus we gain the clear understanding of a network by losing some representational power. However, note that the performance differences of the two networks with the same appropriate number of hidden units are not sig-

nificant.

The results of the NETtalk experiments show that the discrete network is able to find good solutions for such a large problem, and the performance is comparable to that of the back-propagation network, though always a little worse.

Research is under way to interpret the internal representations of discrete networks, and to extract classification rules from trained networks.

VI. CONCLUSION

A pseudo-gradient learning scheme for discrete networks, or multi-layer perceptrons with hard-limiting units is proposed. For the case of single-hidden-layer networks, we showed that the proposed pseudo-gradient always points in the right down hill direction of the error surface. The experiments on different benchmark data sets show that the discrete networks have comparable performance to that of back-propagation networks. A clear understanding of the network is gained by the discrete structure at the cost of some loss of representational power. The preliminary results are encouraging for further study of such discrete networks.

REFERENCES

- [1] R.A. Fisher, "The use of multiple measurements in taxonomic problems," *Annual Eugenics*, 7, Part II, 1936.
- [2] M. Minsky, S. Papert, *Perceptrons*, MIT Press, 1969.
- [3] R. P. Gorman and T. J. Sejnowski, "Analysis of hidden units in a layered network trained to classify sonar targets," *Neural Networks*, Vol. 1, 1988.
- [4] D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, *Parallel Distributed Processing*, MIT Press, 1986.
- [5] T. J. Sejnowski, C. R. Rosenberg, "Parallel networks that learn to pronounce English text," *Complex Systems*, Vol. 1, 1987.
- [6] Z. Zeng, R. Goodman, P. Smyth, "Learning finite state machines with self-clustering recurrent networks," *Neural Computation*, Vol. 5, No. 6, 1993.
- [7] Z. Zeng, R. Goodman, P. Smyth, "Discrete recurrent neural networks for grammatical inference," to appear in *IEEE Transactions on Neural Networks*.